UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Fall 2013**
Doug Stinson **Ryan Henry**

ASSIGNMENT 1

Assignment due date: **Friday, October 4th, 2013 6:00 pm**

Written Response TA: Kevin Henry <k2henry@cs.uwaterloo.ca>
Programming Question TA: Sarah Harvey <sharvey@cs.uwaterloo.ca>
Office hours: Tuesdays, 3:30-4:30pm, DC2102
Total marks: [80 marks]

Written Response Questions [40 marks]

Note: For written questions, please be sure to use complete, grammatically correct sentences where appropriate. You will be marked on the presentation and clarity of your answers as well as the content.

1. (15 marks) Recent security/privacy news has been dominated by documents, leaked by Edward Snowden, that provide an unprecedented look at the cryptanalytic capabilities of the NSA. The latest documents hint at the NSA's ability to circumvent many technologies that were thought to be secure. These findings were published jointly by the New York Times, The Guardian, and Pro Publica. Note that the Pro Publica and Times articles contains links to two of the actual leaked documents, which you should read as well.

   `http://www.propublica.org/article/the-nsas-secret-campaign-to-crack-undermine-internet-encryption`

   `http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html`

   `http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security`

   (a) (2 marks) What is the codename of the current NSA program? What about the similar program run by GCHQ?

   (b) (2 marks) Which countries have access to this program, and what name is given to this group?

   (c) (5 marks) Name at least 5 technologies/protocols targeted by this program.

(d) (2 marks) Is the NSA able to completely break (i.e., recover plaintext messages) from each of these technologies? Provide a single sentence quote from one of the documents that justifies your answer.

(e) (2 marks) What are the “big four” technology companies that are being targeted by GCHQ?

(f) (2 marks) Who is the ”adversary” referred to in the leaked documents?

2. (10 marks) These articles do not contain very specific information on how the NSA breaks security protocols, but the following Wired article discusses one avenue of attack used by the NSA to compromise systems:

   `http://www.wired.com/threatlevel/2013/09/nsa-router-hacking/`

   (a) (3 marks) The textbook describes three components of a computer system: *hardware*, *software*, and *data*. Give an example of each of these components with respect to network routers.

   (b) (4 marks) We have discussed 4 types of attacks against computer systems: *Interception*, *interruption*, *modification*, and *fabrication*. Assume an adversary has gained root access to a router. Given an example of an attack for each of the four types that the adversary could realistically perform.

   (c) (3 marks) The textbook describes the MOM principles for an attack: *Method*, *opportunity*, and *motive*. You can review these definitions using the link at the end of this section.

      i. What does the article speculate the method of attack is?

      ii. Is there a greater opportunity to attack a router instead of a specific user?

      iii. What is the motive for attacking a router instead of a specific user?

3. (15 marks) Bruce Schneier, a well-known security expert, has been involved with handling some of the leaked NSA documents. In the following article he provides 5 suggestions for keeping your communications private in light of the NSA’s newly revealed capabilities.

   `http://www.theguardian.com/world/2013/sep/05/nsa-how-to-remain-secure-surveillance`

   (a) (10 marks) In class we discussed 5 techniques for defending against a threat: *Prevent*, *deter*, *deflect*, *detect*, *recover*. For each of the 5 tips, explain which type of defence it is. If you feel there is more than one acceptable answer, choose the one you feel best matches and provide your justification. Please ensure your answers are numbered with the same numbers from the article.

   (b) (2 marks) Which of the 5 techniques best describes the defense outlined in this article?

      `http://www.theguardian.com/technology/2013/sep/09/nsa-sabotage-dead-mans-switch`

(c) (3 marks) In his article, Bruce Schneier states the following:

"Prefer conventional discrete-log-based systems over elliptic-curve systems; the latter have constants that the NSA influences when they can."

Name a standards organization that has likely been influenced by the NSA. Name a specific elliptic-curve standard that has likely been weakened by the NSA. This question will require a small amount of independent research.

NOTE: See Section 1.2 of the textbook for more information about the above terms.
`http://proquest.safaribooksonline.com/9780132390774/ch01lev1sec2`

Programming Question [40 marks]

Background

You are tasked with testing the security of a custom-developed *file submission application* for your organization. It is known that the application was *very poorly written*, and that in the past, this application had been exploited by some users with the malicious intent of *gaining root privileges*. There is some talk of the application having *four or more vulnerabilities*! As you are the only person in your organization to have a background in computer security, only you can *demonstrate how these vulnerabilities can be exploited* and *document/describe your exploits* so a fix can be made in the future.

Application Description

The application is a very simple program to submit files. It is invoked in the following way:

- `submit <path to file> [message]` : this will copy the file from the current working directory into the submission directory, and append the string "message" to a file called `submit.log` in the user's home directory.

There may be other ways to invoke the program that you are unaware of. Luckily, you have been provided with the source code of the application, `submit.c`, for further analysis.

The executable `submit` is *setuid root*, meaning that whenever `submit` is executed (even by a normal user), it will have the full privileges of *root* instead of the privileges of the normal user. You can check which user you are running as with the command `whoami`.

Testing Environment

To help with your testing, you have been provided with a virtual *user-mode linux* (uml) environment where you can log in and test your exploits. These are located on one of the *ugster* machines. You will receive an email with your account credentials for your designated ugster machine.

Once you have logged into your ugster account, you can run `uml` to start your virtual linux environment. The following logins are useful to keep handy as reference:

- `user` (no password): main login for virtual environment
- `halt` (no password): halts the virtual environment, and returns you to the ugster prompt

The executable `submit` application has been installed to `/usr/local/bin` in the virtual environment, while `/usr/local/src` on the same environment contains `submit.c`. Conveniently, someone seems to have left some shellcode in `shellcode.h` in the same directory.

It is important to note all changes made to the virtual environment will be lost when you halt it. Thus it is important to remember to keep your working files in `/share` on the virtual environment, which maps to `~/uml/share` on the ugster environment.

Rules for exploit execution

- You have to submit four exploit programs to be considered for full credit. **Two of your exploit programs MUST target specific vulnerabilities** (the other two may target other vulnerabilities):
  - buffer overflow vulnerability
  - format string vulnerability
- Each vulnerability may be exploited only in a single exploit program. A single exploit program may exploit more than one vulnerability. If unsure whether two vulnerabilities are different, please contact the Programming TA.
- There is a specific execution procedure for your exploit programs ("*sploits*") when they are tested (i.e. graded) in the virtual environment:
  - Sploits will be run in a **pristine** virtual environment, i.e. you should not expect the presence of any additional files that are not already available
  - Execution will be from a clean `/share` directory on the virtual environment as follows: `./sploitX` (where X=1..4)
  - Sploits must not require any command line parameters
  - Sploits must not expect any user input
  - If your sploit requires additional files, it has to create them itself
- For marking, we will compile your exploit programs in the /share directory in a virtual machine in the following way: `gcc -Wall -ggdb sploitX.c -o sploitX`. You can assume that shellcode.h is available in the /share directory.
- Be polite. After ending up in a root shell, the user invoking your exploit program must still be able to exit the shell, log out, and terminate the virtual machine by logging in as user `halt`. Also, please do not run any cpu-intensive processes for a long time on the ugster machines (see below). None of the exploits should take more than about a minute to finish.
- Give feedback. In case your exploit program might not succeed instantly, keep the user informed of what is going on.

Deliverables

Each sploit is worth 10 points, divided up as follows:

- 7 points for a successfully running exploit that gains a root shell
- 3 points for the description of the identified vulnerability/vulnerabilities, saying how your exploit program exploits it/them, and describing how it/they could be repaired.

A total of four exploits must be submitted to be considered for full credit, including a *buffer overflow* sploit and a *format string* sploit.

What to hand in

All assignment submission takes place on the `student.cs` machines (not ugster or the virtual environments), using the `submit` facility. In particular, log in to the Linux student environment (`linux.student.cs.uwaterloo.ca`), go to the directory that contains your solution, and submit using the following command: `submit cs458 1 .` (dot included). CS 658 students should also use this command and ignore the warning message.

By the **a1 deadline**, you are required to hand in:

sploit1.c, sploit2.c, sploit3.c, sploit4.c: The four completed exploit programs for the programming question.

a1.pdf: A PDF file containing your answers for the written-response questions, and the exploit descriptions for sploit1,2,3,4.

a1.pdf **must contain**, at the top of the first page:

- Your Name
- UW Username
- UW Student Number

You will be deducted 3 marks if any one of these bits of information is missing

Be sure to "embed all fonts" into your PDF files. Some students' files were unreadable in the past; if we can't read it, we can't mark it. We also strongly encourage you to test your exploits in a clean `/share` directory (see earlier).

The 24 hour late policy, as described in the course syllabus, applies to the assignment due date. Submissions within the 24-hour late period incurs a 25% grade deduction. Submissions after the 24-hour late period will automatically be given a grade of 0. There are no exceptions to this policy.

Useful Information For Programming Sploits

Most of the exploit programs do not require much code to be written. Nonetheless, we advise you to start early since you will likely have to read additional information to acquire the necessary knowledge for finding and exploiting a vulnerability. Namely, we suggest that you take a closer look at the following items:

- Module 2
- Smashing the Stack for Fun and Profit (http://insecure.org/stf/smashstack.html)
- Exploiting Format String Vulnerabilities (v1.2) (http://julianor.tripod.com/bc/formatstring-1.2.pdf) (Sections 1-3 only)
- The manpages for dup (man 2 dup), exec (man 3 exec), su (man su)

Note that in the virtual machine you cannot create files that are owned by root in the `/share` directory. Similarly, you cannot run chown on files in this directory. (Think about why these limitations exist.)

GDB

The gdb debugger will be useful for writing some of the exploit programs. It is available in the virtual machine. In case you have never used gdb, you are encouraged to look at a tutorial (e.g.,http://www.unknownroad.com/rtfm/gdbtut/).

Assuming your exploit program invokes the `submit` application using the `execve()` (or a similar) function, the following statements will allow you to debug the `submit` application:

1. `gdb sploitX` (X=1..4)
2. `catch exec` (This will make the debugger stop as soon as the `execve()` function is reached)
3. `run` (Run the exploit program)
4. `symbol-file /usr/local/bin/submit` (We are now in the `submit` application, so we need to load its symbol table)
5. `break main` (Set a breakpoint in the `submit` application)
6. `cont` (Run to breakpoint)

You can store commands 2-6 in a file and use the "`source`" command to execute them. Some other useful gdb commands are:

- "`info frame`" displays information about the current stackframe. Namely, "saved ip" gives you the current return address, as stored on the stack. Under saved registers, eip tells you where on the stack the return address is stored.
- "`info reg esp`" gives you the current value of the stack pointer.
- "`x <address>`" can be used to examine a memory location.
- "`print <variable>`" and "`print &<variable>`" will give you the value and address of a variable, respectively.
- See one of the various gdb cheat sheets (e.g., http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf) for the various formatting options for the print and x command and for other commands.

Note that `submit` will not run with root privileges while you are debugging it with gdb. (Think about why this limitation exists.)

The Ugster Course Computing Environment

In order to responsibly let students learn about security flaws that can be exploited in order to become “root”, we have set up a virtual “user-mode linux” (uml) environment where you can log in and mount your attacks. The gcc version for this environment is the same as described in the article “Smashing the Stack for Fun and Profit”; we have also disabled the stack randomization feature of the 2.6 Linux kernel so as to make your life easier. (But if you’d like an extra challenge, ask us how to turn it back on!)

To access this system, you will need to use ssh to log into your account on one of the `ugster` environment: `ugsterXX.student.cs.uwaterloo.ca`. There are a number of ugster machines, and each student will have an account for one of these machines. You will get an e-mail with your password and telling you which ugster you are to use. If you do not receive a password please check your spam folder. When logged into your ugster account, you can run “`uml`” to start the user-mode linux to boot up a virtual machine.

The gcc compiler installed in the uml environment may be very old and may not fully implement the C99 standard. You might need to declare variables at the beginning of a function, before any other code. You may also be unable to use single-line comments (“//”). If you encounter compile errors, check for these cases before e-mailing the Programming TA.

Any changes that you make in the uml environment are lost when you exit (or upon a crash of user-mode linux). **Lost Forever**. Anything you want to keep must be put in `/share` in the virtual machine. This directory maps to `~/uml/share` on the ugster machines, which is how you can copy files in and out of the virtual machine. It is wisest to ssh twice into ugster. In one shell, start user-mode linux, and compile and execute your exploits. In the other account, log into ugster and edit your files directly in `~/uml/share/`, so as to ensure you do not lose any work. The ugster machines are not backed up. You should copy all your work over to your student.cs account regularly.

When you want to exit the virtual machine, use exit. Then at the login prompt, login as user “halt” and no password to halt the machine.

Any questions about your ugster environment should be directed towards the Programming TA.